

FINAL REPORT

LINC EVALUATION PROGRAM

L. HUNDLEY

FACILITY FORM 602
 N 66-87574
 (ACCESSION NUMBER)
 67
 (PAGES)
 CR-6-8609
 (NASA CR OR TMX OR AD NUMBER)
 (THRU)
 Noxe
 (CODE)
 (CATEGORY)



INSTRUMENTATION RESEARCH LABORATORY, DEPARTMENT OF GENETICS
STANFORD UNIVERSITY SCHOOL OF MEDICINE
PALO ALTO, CALIFORNIA

~~_____~~
~~_____~~
~~_____~~
~~_____~~

FINAL REPORT

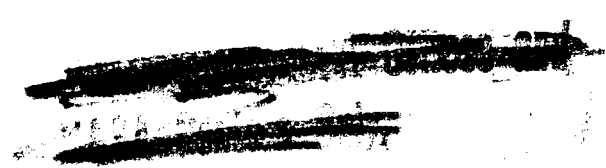
LINC Evaluation Program

J. Lederberg

L. Hundley

Department of Genetics
Stanford University

March 1965



FORWARD

This report is a final report submitted to the United States Department of Health, Education, and Welfare in connection with their LINC Computer Evaluation Program. Under their Grant No. FR 00151-01, the Instrumentation Research Laboratory was supplied with a LINC Computer and some additional accessory equipment. The applications of this computer and its evaluation represent work carried out under National Aeronautics and Space Administration Grant NsG 81-60. For this reason, this report is being submitted both to the LINC Evaluation Board in fulfillment of their requirement for a final report and to NASA as a technical report.

CONTENTS

- I. Introduction
- II. General Use I-O Equipment and Programs
- III. Utility Programs
- IV. Experiment-Related Programs and Hardware
- V. The LINC Evaluation Program as a Training Technique
- VI. Computer Performance
- VII. Conclusion
- Appendix A: Selected Programs
- Appendix B: Log Book

I. INTRODUCTION

The instrumentation Research Laboratory within the Department of Genetics has as its purpose the design of special purpose instruments for biological research. This includes electrical, mechanical and optical design. The LINC in our laboratory has been used as a system element in a number of experimental situations and its use has proven to be both education to us and experimentally rewarding.

Headed by Dr. Joshua Lederberg and under the direction of Dr. Elliot Levinthal, the laboratory has as its primary mission the development of life detection systems on a microbial level for remote Martian exploration. In order to accomplish this end, a number of different types of physical measurements have been investigated in great detail. We believe that these studies, a number of which involve LINC, will also result in new instrumentation and techniques of general laboratory utility.

We wish to request that the LINC be permanently assigned to our laboratory.

II. GENERAL USAGE I-O EQUIPMENT AND PROGRAMS

Our LINC has been equipped with a number of peripheral devices. These include a Datamec IBM compatible tape recorder, a Calcomp plotter, and a teletype. In the process of being installed is a 4096 word external memory.

The Datamec is equipped for two speed (45 and 4.5 ips) two density (200 and 566 bpi) operation, with both read and write capability. These speeds and densities give us a wide range of data rates. The upper limit is 25,000 six bit characters per second. The interface is very simple and required only two cards. One of these would be eliminated if the gated accumulator lines were being used for nothing else.

Programs for the Datamec include those to read and write IBM compatible format, generate data tapes from continuous on-line input, and to regroup the input data on LINC tape blocks and then, if desired, to rewrite these blocks into IBM format. All of these combinations form a highly flexible system. Use of the Datamec has completely superseded the IBM 026 key-punch.

The Calcomp plotter has been in operation for some time now and has proven to be extremely useful. Programs for plotting all forms of data have been written. These include both ordinate and abscissa scaling and linear interpolation. A program has also been written for character generation which includes character size scaling and positioning.

The teletype has proven to be a very good means of getting both program and data into LINC and getting hard copy of both out. Its major drawbacks are its low speed, lack of tabs and that it is somewhat noisy; however, we know of no cheaper means of getting printed output. Input and output routines have been written which calculate teletype code from LAP code and viseversa which take about twenty locations each, so memory usage is not excessive.

The 4096 word memory, which should be in operation within the next few weeks, will be used both for program and data handling. There will be three modes of operation which are: 256 word input and output gulps at

eight microseconds per word and single word input which indexes the memory address register with each input. This later mode is designed mainly for data handling.

III. UTILITY PROGRAMS

These programs include those for program input, assembly, and debugging, for keyboard data input and computation and for data display. Most of the programs to be mentioned are more completely described in Appendix A.

The LINCT system is our teletype program text input-output system which has a number of useful features. It is tied into a modified LAP which will assemble for the 2K memory.

We have operating on the IBM 7090 a compiler for LINC which uses a modified Balgol language. This system, called "BLINC", and a program operating system which was written in BLINC are described in some detail in the appendix.

Debugging routines include an octal to Mnemonic converter and print-out program, and a program which follows another program through all of its branching to determine which locations contain instructions and which contain constants. This is used with the converter program to get a proper print-out. A print-out of LAP III was obtained in this way.

A Floating point package with two word mantissa has been written. Copies of this program and a usage explanation will be available shortly. This program has been incorporated into a desk calculator with storage routine. This routine has the usual arithmetic operations as well as square root, e^X , $\log_e X$, $\sin X$, $\cos X$, and 2×2 Chi square. It is arranged for the easy addition of other arithmetic subroutines. Teletype input and output and certain manipulations of the stored data are included.

A number of simple algebraic programs have been written, such as those for mean and standard deviation, Chi square and other statistical operations.

Display programs include those for point and bar graph display with X and λ scaling keyboard calling of data sets. These data sets may be

manipulated in a number of ways including inversion, addition, multiplication and rotation.

These are the major programs of a general usage nature now in operation. The only major programming effort now being considered in this class is a simple arithmetic compiler based on the two word floating point system. A more complete symbolic compiler is a possibility, but due to the large amount of effort involved will probably not be undertaken for some time.

IV. EXPERIMENT RELATED PROGRAMS AND HARDWARE

Most of the research in our department is involved with experimentation either on a bacterial or molecular level; therefore all of the on-line LINC experiments that have been done have involved physical methods such as mass spectroscopy, radioactive and fluorescent tagging, fluorescent decay times and particle counting. An anticipated experiment involves the interpretation of Raman spectra.

The LINC has been directly connected to the output of the Bendix time-of-flight mass spectrometer. Output from the mass spectrometer is reduced as it comes into LINC into mass amplitude and time of occurrence. The direct determination of mass number is difficult due to instability in the Bendix's scanning ramp. One means of overcoming this, which will be tried, is to allow LINC to generate the scanning ramp by the use of a mechanical D-A converter which has been built in our shop. This consists of a 200 step per revolution stepping motor driving a ten turn pot. This is a very simple system and has proven most useful. This use of LINC ties in with a much larger system which is a computer program for the direct determination of compound composition from mass spectra. This work is being done under a separate grant and the initial program is being run on the IBM 7090 at the Stanford Computation Center.

The LINC has been used in a number of ways in experiments with fluorescent compounds. The first experiment of this type used LINC as modulator, phase locked detector, and integrator in an extremely sensitive fluorometer. With integration times of ten minutes, the detection of 10^{-13}

molar solutions of fluorescein with a signal to noise of 15 to 1 were obtained using a 400 milliwatt light source. This experiment was performed to determine parameters for a sensitive fluoremeter as part of our effort to design apparatus for the detection of life on Mars. A program is now being written which will determine the best fluorescent system transfer function for a given material by generating all possible combinations of filters, light sources, and phototubes. The data for the components of this system will be stored as sets on LINC tape.

A system has been built for the determination of fluorescent decay times in the low nanosecond region. This consists of a fast flash lamp, photomultiplier tube, sampling scope and LINC as a 512 channel integrator. Calculation shows that we will get about two quanta per channel per flash. Our design goal is to investigate materials with decay times on the order of five nanoseconds. To date, our best results have been in the 10 nanosecond region. The limiting factor is the lamp decay time. This will be improved by the use of a different type of lamp. The LINC has performed most admirably in this application. No external hardware was required except the mechanical D-A convertor for driving the sampling scope sweep. This experiment is being conducted in cooperation with Dr. Lubert Stryer of the Stanford Biochemistry Department. A program will be written to get a best exponential fit to the experimental data so that direct time constant output will be available.

Programs have been written for the keyboard input of data from nuclear counters which determine mean and standard deviations as well as sorting data sets according to size distributions and normalizing the data. These programs have been in routine use by a group in the Genetics Department under the direction of Dr. Leonard Herzenberg. This group is studying antibody reactions in mice.

These programs have, by the rapid presentation of results, allowed the experimenters to determine what the next step in their procedure should be with very little delay, and has therefore increased the number of experiments which they are able to perform by a factor of two to three.

V. THE LINC EVALUATION PROGRAM AS A TRAINING TECHNIQUE.

In general, the experience gained with digital techniques has been of great value to all of us here. The instruction initially received on LINC was quite adequate with one exception. It would have been very desirable to spend more time on use and misuse of the various I-O functions. It has been in this area that most of our nonproductive time has been spent. From the overall point of view, LINC has been a most demanding teacher in its own right. It has changed and simplified our approach to many problems. It has also made possible experiments which would otherwise have been too time consuming to perform.

Several undergraduate and medical students have gained proficiency in systems programming on LINC. It is an excellent machine from the standpoint of man-machine interaction but higher level languages would give a more realistic interaction to sophisticated systems.

VI. COMPUTER PERFORMANCE

The performance of LINC in respect to maintenance has far exceeded reasonable expectation. After approximately 3200 hours of operation, the only failures have been one bad cable connection and two output transistors whose failure can be traced to external misuse.

The general performance of LINC in the laboratory has been entirely adequate and most rewarding. Most of the recommendations that come to mind must be admitted to be generated by our own special requirements; however, there are three recommendations which it is felt are of general interest to most users.

The first area is that of multiple word arithmetic. Any instruction changes which would reduce program length and running time would be a great help. These might include clearing the accumulator on a LAM instruction and recovery of both halves of a multiply.

The second suggestion is to make all of the 2K memory programable. This would be very useful when performing complex computations and

would reduce the running time of a number of programs which we now operate by minimizing the number of tape transfers involved. A suggested means of achieving this is being transmitted under separate cover to S.M. Orinsten at the Computer Research Laboratory.

Our third point is that a problem-oriented compiler (e.g. artran or Atyol) would be extremely useful. Even if the compilation were somewhat slow, the reduction in programing time should still be very large. Mnemonic print-outs of the compiled program can allow the programmer to see exactly what is happening and give him a framework in which to get machine code zonations.

VII. CONCLUSIONS

The concept of what an ideal laboratory computer should be will vary greatly among various investigators. From our point of view, LINC has proven to be a very useful system. The careful attention of the designers to those points which are most important for the on-line use of a computer is obvious and most gratifying.

It has become apparent that in the future we will want to have on-line computer capability even greater than that provided by LINC. Greater word length, higher A-D resolution, larger memory, greater speed and smaller physical size will be the types of improvements that we will be looking for in new machines. A system such as IBM's 1800 is a step in the right direction. This desire for a larger capability has certainly been the result of the use of LINC itself. We feel that future developments must proceed in this direction if full advantage is to be taken of the experience gained from the LINC program.

Appendix: A

Selected Program Discriptions

Double Precision Floating Point

Winter 1965

t. coburn

General Information

1. A double precision floating point word consists of three 12-bit words in the following sequence: exponent, high order word, low order word. The last two of these are collectively called the "mantissa".
2. Exponent and mantissa each contain a sign in the leftmost bit, i.e. the 11 bit of the exponent or 23 bit of the mantissa.
3. The mantissa is a fraction between +1 and -1; that is, the decimal point is assumed to be at the left of bit 22.
4. The mantissa is left adjusted. This means that except for zero words, all positive mantissas will contain a 1 in bit 22, and all negative words will contain a zero in bit 22.
5. Integers can and indeed must be used for some of the routines available. These are automatically floated before they are used.
6. A floating point accumulator(FAC) is maintained in locations 1120, 1121, and 1122. It is used in the same way that the regular accumulator is used.
7. The other half of any operation is called the operand or argument.
8. The address of a double precision floating point word is the location of the exponent. Integers are addressed as usual.
9. The floating point routines use index registers 12-17. These registers are not restored on leaving the floating point package.
10. Entrance to the floating point package is accomplished by jumping to a three instruction routine located some place in core (see next page).
11. After the last operation code the program exits and continues executing regular Linc instructions.
12. There is no rounding off within the floating point package.

Instructions for using the package

The following sequence of instructions will serve as an example of the necessary format.

176	:			
177	:			
200	Jmp	375 -----	375	Lda
201	0400	(operand address)	376	0
202	4001	(operation code)	377	Jmp 1000
203	0403	(operand address)		
204	4002	(operation code)		
205	0400	(operand address)		
206	0023	(operation code)		
207	:			
210	:			

Operand Address

This may be a direct address: 400
or an indirect address: 4002
or it may be zero.

1. In a direct address the location, 400, contains the exponent of the floating point word, or an integer as the case may be.
2. In an indirect address the index register, 2, refers to the corresponding address. Bit 12, the 4000, bit signifies that the address is indirect.
3. A zero operand refers to the floating point accumulator. Hence, to square a number in the FAC, one executes a multiply specifying a zero operand.

Operations

1. Operations available are listed in the following table.
2. The 4000 bit in the operation code is used to indicate whether this operation is the last in a series. In the example above, if the next location following the code 0023 contained 0400, this would be interpreted as "sxl". If the last code had been 4023, then the next location would be the address of an operand.
3. Some operations, fix and sign, are meaningless unless they are the last in a series since the result is left in the regular accumulator.

Table of codes and operations

<u>Code</u>		<u>Operation</u>
1	Cla	Clear and add operand to FAC.
2	Add	Add a floating point word to FAC.
3	Com	Complement operand; leave in FAC.
4.	Mul	Multiply FAC times floating point word.
5	FAC/OP	Divide Fac by floating point word.
6	OP/FAC	Divide operand by FAC, result in FAC.
7	I+FAC	Add an integer to FAC
10	IxFAC	Multiply FAC by an integer.
11	FAC/I	Divide FAC by an integer.
12	I/FAC	Divide integer by FAC, result in FAC.
13	Fix	Convert a floating point word to an integer. Result is left in regular accumulator.
14	Flt	Float an integer, result in FAC.
15	Clr	Zero put in operand and FAC.
16	Max	Compare size of operand with FAC. Larger left in FAC.
17	Min	" " " Smaller " " .
20	SGn	If operand is less than zero, -1 is left in regular acc. If operand equals zero, 0 is left in regular acc. If operand is greater than zero, +1 is left in reg. acc.
21	incr	Increment operand by FAC, leave in Fac as well. This is equivalent to an add to memory.
22	Sub	Subtract operand from FAC. Result left in FAC.
23	Sto	Store FAC in address of operand. Leave in FAC.
24	SSP	Set sign of operand plus; i.e. complement if negative.
25	SSM	Set sign of operand minus; " " " " positive.

1000 ADA+
 1001 1776
 1002 STC 17
 1003 SET+13
 1004 1121
 1005 SFT+14
 1006 1122
 1007 SFT+16
 1010 1125
 1011 SET+15
 1012 1124
 1013 LDA+17
 1014 AZE
 1015 JMP 1020
 1016 ~~JMP 1613~~ LDA+
 1017 ~~JMP 1037~~ 1120
 1020 APO+
 1021 JMP 1026
 1022 BCO+
 1023 6000
 1024 STC 1025
 1025 HLT
 1026 STA+
 1027 0
 1030 STC 12
 1031 LDA 12
 1032 STC 1123
 1033 LDA+12
 1034 STC 1124
 1035 LDA+12
 1036 STC 1125
 1037 CLR
 1040 LDA+17
 1041 BCL+
 1042 4000
 1043 ADA+
 1044 1072
 1045 STC 12
 1046 LDA 12
 1047 STC 1050
 1050 HLT
 1051 STC 1062
 1052 LDA 17
 1053 APO
 1054 JMP 1013
 1055 LDA+
 1056 6001
 1057 ADD 17
 1060 STC 1063
 1061 LDA+
 1062 0
 1063 HLT
 1064 JMP 1450
 1065 JMP 1441
 1066 JMP 1051
 1067 JMP 1450
 1070 JMP 1321
 1071 JMP 1441
 1072 JMP 1051
 1073 JMP 1071 1)
 1074 JMP 1177 2)
 1075 JMP 1543 3)
 1076 JMP 1252 4)
 1077 JMP 1525 5)

Pick up operand
and operation

Jump to specified operation

Check for return

Float Arg and return

Integer Arg / FAC

Jump table

1100 JMP 1321 6)
 1101 JMP 1531 7)
 1102 JMP 1534 10)
 1103 JMP 1536 15)
 1104 JMP 1067 12)
 1105 JMP 1135 13)
 1106 JMP 1064 14)
 1107 JMP 1653 15)
 1110 JMP 1557 16)
 1111 JMP 1546 17)
 1112 JMP 1555 20)
 1113 JMP 1133 21)
 1114 JMP 1774 22)
 1115 JMP 1137 23)
 1116 JMP 1571 24)
 1117 JMP 1574 25)

Jump table

1120 0
 1121 0
 1122 0
 1123 0
 1124 0
 1125 0
 1126 0
 1127 0
 1130 0
 1131 0
 1132 0

Location of Floating point accumulator (FAC).

Location of register A, or Argument (ARG).

Location of register B.

Register Q, used in divide.

1133 JMP 1177
 1134 JMP 1137
 1135 JMP 1441
 1136 JMP 1474
 1137 SET 12

increment

Fix

1140 1027
 1141 LDA
 1142 1120
 1143 STA 12
 1144 LDA 13
 1145 STA 12
 1146 LDA 14
 1147 STA 12
 1150 JMP 1051

Store and return

1151 SET 12
 1152 0
 1153 LDA
 1154 1120
 1155 COM
 1156 ADD 1123
 1157 AZE
 1160 JMP 1166
 1161 APO
 1162 XSK 12
 1163 JMP 12
 1164 SET 12
 1165 0
 1166 LDA 13
 1167 COM
 1170 ADA 15
 1171 AZE
 1172 JMP 1161
 1173 LDA 14
 1174 COM
 1175 ADA 16
 1176 JMP 1161
 1177 LDA

Which is larger? FAC or Arg.

Begin Add

1201 STC 1241
1202 JMP 1424
1203 JMP 1600
1204 STC 1231
1205 ADD 1120
1206 COM
1207 ADD 1123
1210 AZE†
1211 JMP 1227
1212 APO†
1213 JMP 1221
1214 STC 12
1215 JMP 1604
1216 JMP 1613
1217 JMP 1622
1220 JMP 1223
1221 COM
1222 STC 12
1223 JMP 1633
1224 XSK† 12
1225 JMP 1223
1226 NOP
1227 JMP 1666
1230 SRQ†
1231 0
1232 JMP 1242
1233 JMP 1600
1234 APO†
1235 JMP 1241
1236 JMP 1633
1237 JMP 1735
1240 NOP
1241 HLT
1242 JMP 1700
1243 JMP 1246
1244 JMP 1751
1245 JMP 1241
1246 JMP 1712
1247 JMP 1241
1250 JMP 1721
1251 JMP 1246
1252 JMP 1600
1253 STC 1316
1254 ADD 1123
1255 ADD 1120
1256 STC 1120
1257 JMP 1756
1260 JMP 1432
1261 JMP 1751
1262 SET† 12
1263 7763
1264 CLR
1265 S50
1266 1130
1267 JMP 1666
1270 LDA 13
1271 ROR† 1
1272 JMP 1641
1273 XSK† 12
1274 JMP 1264
1275 SET† 12
1276 7764
1277 CLR

Add Arg to FAC

Multiply Arg x FAC

1300 SRQ
 1301 1127
 1302 JMP 1666
 1303 LDA 13
 1304 ROR+1
 1305 JMP 1641
 1306 XSK+12
 1307 JMP 1277
 1310 ADD 1126
 1311 STC 1120
 1312 JMP 1712
 1313 JMP 1315
 1314 JMP 1721
 1315 SRQ+
 1316 0
 1317 JMP 1742
 1320 JMP 1051
 1321 JMP 1600
 1322 STC 1421
 1323 JMP 1756
 1324 JMP 1700
 1325 JMP 1330
 1326 JMP 1751
 1327 JMP 1051
 1330 JMP 1164
 1331 JMP 1340
 1332 JMP 1432
 1333 JMP 1441
 1334 JMP 1633
 1335 NOP
 1336 JMP 1613
 1337 JMP 1341
 1340 JMP 1432
 1341 ADD 1126
 1342 COM
 1343 ADD 1123
 1344 ADD 1665
 1345 STC 1126
 1346 STC 1131
 1347 STC 1132
 1350 SET+12
 1351 1131
 1352 JMP 1622
 1353 JMP 1742
 1354 JMP 1356
 1355 JMP 1622
 1356 JMP 1666
 1357 JMP 1666
 1360 JMP 1613
 1361 LDA 15
 1362 APC+
 1363 JMP 1366
 1364 JMP 1700
 1365 JMP 1376
 1366 LDA 12
 1367 RCO+
 1370 4000
 1371 STA 12
 1372 SRQ
 1373 1370
 1374 JMP 1402
 1375 JMP 1352
 1376 SRQ

Multiply (continued)

Divide Arg by FAC

1400 JMP 1402
1401 JMP 1355
1402 XSK:12
1403 SRC:
1404 2525
1405 JMP 0
1406 CLR
1407 ADD 1132
1410 STC 1122
1411 ADD 1131
1412 STA 13
1413 APO:
1414 JMP 1420
1415 JMP 1633
1416 JMP 1735
1417 NOP
1420 SRC:
1421 0
1422 JMP 1742
1423 JMP 1651
1424 JMP 1700
1425 JMP 1427
1426 JMP 1671
1427 JMP 1604
1430 JMP 1613
1431 JMP 1646
1432 LDA 13
1433 STC 1127
1434 LDA 14
1435 STC 1130
1436 ADD 1120
1437 STC 1126
1440 JMP 0
1441 LDA 15
1442 STC 1121
1443 LDA 16
1444 STC 1122
1445 ADD 1123
1446 STC 1120
1447 JMP 0
1450 LDA
1451 0
1452 STC 1467
1453 JMP 1432
1454 ADD 1123
1455 STA 13
1456 SCR 13
1457 STC 1122
1460 ADD 1632
1461 STC 1120
1462 JMP 1700
1463 JMP 1470
1464 JMP 1751
1465 JMP 1613
1466 JMP 1622
1467 FLT
1470 JMP 1712
1471 JMP 1465
1472 JMP 1721
1473 JMP 1470
1474 LDA
1475 1120
1476 AZE
1477 APO

Divide (continued)

Check for zero Arg'm Add.

Copy FAC into B

Copy Arg into FAC

Float Arg

Fix FAC

1501 ADA
 1502 7764
 1503 APO
 1504 JMP 1517
 1505 COM
 1506 ADA
 1507 340
 1510 STC 1512
 1511 ADD 1121
 1512 HLT
 1513 JMP 1051
 1514 LDA 13
 1515 SCR 13
 1516 JMP 1051
 1517 LDA 13
 1520 SCR 13
 1521 APO
 1522 ADD 1720
 1523 ADD 1664
 1524 JMP 1051
 1525 JMP 1604
 1526 JMP 1613
 1527 JMP 1622
 1530 JMP 1321
 1531 JMP 1450
 1532 JMP 1177
 1533 JMP 1051
 1534 JMP 1450
 1535 JMP 1252
 1536 JMP 1450
 1537 JMP 1604
 1540 JMP 1613
 1541 JMP 1622
 1542 JMP 1321
 1543 JMP 1441
 1544 JMP 1742
 1545 JMP 1051
 1546 JMP 1151
 1547 JMP 1441
 1550 JMP 1051
 1551 JMP 1151
 1552 JMP 1554
 1553 JMP 1441
 1554 JMP 1051
 1555 JMP 1441
 1556 JMP 1700
 1557 JMP 1562
 1560 CLR
 1561 JMP 1051
 1562 LDA 13
 1563 SCR 13
 1564 APO
 1565 ADD 1570
 1566 ADD 1720
 1567 JMP 1051
 1570 ~~JMP~~ 7775
 1571 JMP 1756
 1572 JMP 1441
 1573 JMP 1051
 1574 JMP 1756
 1575 JMP 1441
 1576 JMP 1742
 1577 JMP 1051

Fix FAC (continued)

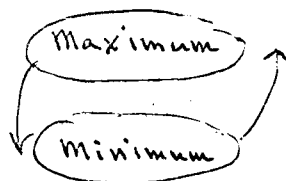
FAC / Arg

Integer + FAC

Integer x FAC

FAC / Integer

Complement



Sign

Set sign plus

Set sign minus

1600 LDA 15
 1601 RCG 13
 1602 SCR 13
 1603 JMP 0
 1604 LDA 16
 1605 STC 1130
 1606 ADD 1124
 1607 STC 1127
 1610 ADD 1123
 1611 STC 1126
 1612 JMP 0
 1613 LDA 14
 1614 STC 1125
 1615 ADD 1121
 1616 STC 1124
 1617 ADD 1120
 1620 STC 1123
 1621 JMP 0
 1622 CLR
 1623 ADD 1130
 1624 STC 1122
 1625 ADD 1127
 1626 STC 1121
 1627 ADD 1126
 1630 STC 1120
 1631 JMP 0
 1632 13
 1633 CLR
 1634 ADD 1720
 1635 ADD 1120
 1636 STC 1120
 1637 LDA 13
 1640 SCR 1
 1641 STC 1121
 1642 ADD 1122
 1643 RCR 1
 1644 STC 1122
 1645 JMP 0
 1646 JMP 1622
 1647 JMP 1700
 1650 JMP 1203
 1651 JMP 1441
 1652 JMP 1241
 1653 JMP 1751
 1654 JMP 1137
 1655
 1656
 1657
 1660
 1661
 1662
 1663 0
 1664 3777
 1665 7776
 1666 CLR
 1667 LDA 16
 1670 LAM 14
 1671 LDA 15
 1672 LAM 13
 1673 STC 1062
 1674 LAM 14
 1675 STC 1062
 1676 LAM 13
 1677 JMP 0

? sign of FAC = Arg?

Copy Arg into B

Copy FAC into Arg

Copy B into FAC

Scale FAC right

Check for zero in Add.

Clear

Not used

Add mantissa of Arg to FAC.

LDA 13
 1701 AZE
 1702 JMP 0
 1703 SCR 1
 1704 LDA 14
 1705 ROR 1
 1706 AZE
 1707 JMP 0
 1710 XSK
 1711 JMP 0
 1712 LDA 13
 1713 ROL 1
 1714 BCO 13
 1715 APC
 1716 XSK
 1717 JMP 0
 1720 1
 1721 CLR
 1722 ADD 1665
 1723 ADD 1120
 1724 STC 1120
 1725 LDA 14
 1726 ROL 1
 1727 STA 14
 1730 LDA 13
 1731 ROL 1
 1732 STC 1121
 1733 LAM 14
 1734 JMP 0
 1735 LDA 13
 1736 BCO
 1737 4000
 1740 STC 1121
 1741 JMP 0
 1742 LDA 13
 1743 COM
 1744 STC 1121
 1745 ADD 1122
 1746 COM
 1747 STC 1122
 1750 JMP 0
 1751 CLR
 1752 STC 1120
 1753 STC 1121
 1754 STC 1122
 1755 JMP 0
 1756 SET 12
 1757 0
 1760 LDA 13
 1761 APC
 1762 JMP 1743
 1763 LDA 15
 1764 APC
 1765 JMP 12
 1766 COM
 1767 STC 1124
 1770 ADD 1125
 1771 COM
 1772 STC 1125
 1773 JMP 12
 1774 JMP 1742
 1775 JMP 1177
 1776 JMP 1742
 1777 JMP 1051

Does FAC = 0 ?

9/2 FAC normalised ?

scale FAC left.

correct sign bit.

Complement FAC

Clear FAC

Complement Arg or FAC if negative.

Subtract

A Preliminary Description of an
Operating System for the LINC Computer

by

Richard K. Moore
Department of Genetics
Stanford University

March 1965

The LINC

The LINC is a binary, 12-bit, 2048-word, digital computer. The core is divided into eight "quarters", each quarter being 256 words long. Quarter 0 contains cells 0 through 377*, quarter 1 contains cells 400 through 777, etc. LINC tapes are divided into 1000 blocks, each block being 400 words long. Thus, for example, the tape read instruction replaces the contents of one quarter of LINC memory with the contents of one tape block. There are two tape units on the LINC, units 0 and 1.

The LINC Operating System: LOSS

LOSS is based on a highly structured use of both tape units as well as the various quarters of memory. LOSS strives to provide a framework in which communication among LINC programs is very simple, thus allowing complex operations, such as compiling, to be accomplished through the successive efforts of relatively simple programs. This framework of simple communication is based on three artifacts: (1) the RECURSIVE OVERLAY, (2) the BUFFER, and (3) TEXTS.

The RECURSIVE OVERLAY

Under LOSS, tape unit 0 is reserved for a program stack and an overlay stack. The program stack consists of those programs which are available to be run on the LINC under LOSS. Each of the programs in the stack is identified by a number. Program 1 occupies blocks 11 through 15,

* From now on, unless otherwise noted, numbers are in the octal system.

program 2 occupies blocks 21 through 25, etc. The overlay stack begins following the top of the program stack. During execution, a program occupies only quarters 1 through 5. Quarter 0 contains a number of routines which are used by LINC programs. One of these routines is an overlay procedure which allows any program in the program stack to be called like a subroutine. As an example, suppose that there are four programs on the program stack and that program 3, currently under execution, wishes to call program 2. In this case program 3 merely places the number 2 in the accumulator and transfers control to the overlay routine. The overlay routine (1) saves the address from which it was called, (2) writes quarters 1 through 5 on tape (at the top of the overlay stack, say blocks 61 through 65), (3) reads program 2 (blocks 21 through 25) into core and (4) begins execution at cell 400. When program 2 has completed its execution (which may have included overlays of other programs, in which case program 2 would have been written on blocks 71 through 75) it merely returns control to the overlay routine which reads back in program 3 from the overlay stack and returns control to the cell saved at step (1) above.

The power of this system is shown by the fact that in the preceding example program 2 was able to perform its function without knowing what program had called it nor in what depth of recursion the overlay process was currently involved.

The BUFFER

While the elements of mathematical or logical operations are variables or arrays (single cells or blocks of consecutive cells), the elements of such operations as input-output or of inter-program communications are

LISTS, where the elements of a LIST are either variables or alphanumeric STRINGS. In order to allow for the manipulation of such LISTS, LOSS includes a general purpose BUFFER (beginning at cell 3000) together with two procedures, PUT and GET, located in quarter 0. The BUFFER is a pushdown stack whose unit of storage is a RECORD. The arguments to the PUT procedure are one or more LISTS; these LISTS are combined by PUT to form a RECORD which is placed on top of the BUFFER. Similarly GET(\$\$L) causes the variables of the LIST L to assume the values found in the top RECORD of the BUFFER, which RECORD is then erased from the BUFFER. Since an OVERLAY affects only quarters 1 through 5, the canonical way for programs to transmit parameters to each other is by means of the BUFFER.

TEXTS

LOSS reserves tape unit 1 for the storage of TEXTS. A TEXT is a group of consecutive tape blocks preceded by a few special code words and a five character name. TEXTS are grouped together to form BOOKS, each BOOK being 100 tape blocks long. Thus BOOK 0 comprises blocks 0 through 77, BOOK 1 comprises blocks 100 through 177, etc. The 0'th block of each BOOK is an index which contains the names of all the TEXTS (in alphabetical order) in its BOOK together with their size and initial blocks. This formalism is not meant to restrict the kind of information which can be stored on tape, but rather makes it possible for allocation of tape storage space to become automatic and somewhat resistant to destructive over-writes. At the same time a block number together with the first two letters of its name become a concise, as well as a securely redundant, way to refer to TEXTS.

The LINC MONITOR

The LINC MONITOR is, by convention, program 1 on the program stack. Its only capabilities are to accept instructions from the typewriter, to perform simple operations upon the buffer, and to overlay any of the programs on the program stack. Its specific operations are:

DISPLAY n	n an octal integer; the n'th RECORD of the BUFFER is displayed on the scope.
TYPE n	The n'th RECORD is typed.
EXECUTE n	The n'th program on the stack is overlayed.
ERASE n	n RECORDS are erased from the top of the BUFFER.
PUT(L)	The LIST L is placed on the BUFFER. L consists of octal integers and alphanumeric STRINGS. A STRING, in this sense, begins with the character " and is terminated by % .

Since LOSS itself includes a method for referring to and loading programs, i.e., the OVERLAY, the MONITOR does not require a "loader" or a list of available "systems".

Richard Moore
March 8, 1965

BLINK

General Description

BLINK is a version of Subalgol designed for use with the LINC computer. Programs very similar to Subalgol programs are translated on the 7090 by the BLINK compiler (which is written in Subalgol), into relocatable LINC code.

Reserved Word Changes with Semantics

BLINK has no "library procedures", though it retains all of Subalgol's "intrinsic functions". The following Subalgol reserved words are without special meaning in BLINK:

STOP, SHLT, SHRT, EXTR, STATEMENT, WHILE, SEGMENT, MONITOR, STEP,
INPUT, OUTPUT, TRACE, DPRECISION, LIBRARY, CARDREAD, PRINTOUT,
COMPLEX, RE, IM, WRITE, READ, SQRT, LOG, EXP, SIN, COS, TAN,
ENTIRE, SINH, COSH, TANH, ARCTAN, ROMXX, ARCSIN, ARCCOS, RCARD,
READM, WRITEM, CHECKM, MOVEM, MOVEFILE, ENDFILE, REWIND, UNLOAD,
FLAGM, etc.

The following reserved words are introduced or redefined with BLINK:

I.

- | | |
|---|-------------|
| A. ROTL, ROTR, SCLR | H. INCR. |
| B. BTCLR, STCOM, STSET | I. OVERLAY |
| C. LDA | J. STRING |
| D. STA | K. LIST |
| E. DO | L. PUT, GET |
| F. REPEAT | M. RESTART |
| G. RDC, RCG, MTB, WRC,
WCG, WRI, CHK, RDE, | N. *GETCOR |
| | O. EXTERNAL |

II.

- A. I1, I2, I7
- B. M, MF, MI
- C. I11, I2I, I7I
- D. POINTER

Corresponding Semantics

I.

- A. ROTL(N,OPERAND) : Intrinsic function; arguments type integer; result type integer; corresponds to ROL instruction; as in later intrinsic functions, the effect of the i-bit is obtained by using a value of $N > 17$.
- B. BTCLR(MASK,OPERAND) : Intrinsic function; types integer; corresponds to BCL, etc.; as with ROTL class function, a constant first argument naturally reduces length of resulting code.
- C. LDA(<arbitrary arithmetic expression>) : Expression is calculated and placed in accumulator; if of type floating, it is truncated to an integer; useful in connection with DO and STA as mentioned below.
- D. STA(<simple variable>) : Contents of the accumulator are placed in the simple variable.
- E. DO(<integer arithmetic expression>) : Expression is evaluated, treated as a LINC instruction and executed, e.g.,*

```

LDA(I)$
DO("470")$ COMMENT AZE1$
GO TO L$
STA(J)$

```

* Within BLINK examples, numbers are decimal unless placed in double quotes.

is identical in effect to:

```
EITHER IF I EQL O$ GO TO L$
```

```
OTHERWISE$ J=I$
```

The DO function, however, is of only dubious value as a tool to create tight code; its real purpose is to allow the use of external device communication instructions in BLINK programs, e.g., OPR, SNS, etc.

- F. REPEAT(<integer expression>)\$ <statement>\$: Identical in effect to:

```
DMY1=<integer expression>$
```

```
FOR DMY2=(1,1,DMY1)$ <statement>$
```

except that the REPEAT loop is more efficient and does not change the value of any variable in its indexing.

- G. RDC(i,u,QNMBR,BNMBR) : Identical to LAP, except that i and u are represented by a 0 or 1.

- H. INCR(<expression>,<variable>) : Identical in effect to:

```
<variable>=<variable>+<expression>
```

except that if the variable is subscripted, INCR calculates the subscript only once and INCR is a function having the new value of <variable> as its value.

- I. OVERLAY<integer expression> : The OVERLAY* routine is entered with the integer expression in the accumulator.

- J. STRING<identifier>(<integer>)=(<alpha string>) : The STRING declaration is identical to the ARRAY declaration, except that only a single dimension, and no irregular subscript ranges, are allowed. The effect of the STRING declaration is different in

* The reader should be familiar with LOSS at this point.

that the zero'th position of the STRING (even though not requested in the declaration) is reserved and filled with the size of the STRING, this information being necessary to the PUT and GET routines. STRINGS may be manipulated word by word, as are ARRAYS, through subscription. Thus S(1) refers to the first and second characters of the STRING S.

- K. LIST : The LIST declaration is identical to the Subalgol OUTPUT declaration except that a STRING name (followed by empty parenthesis is allowed as a LIST element, and fulfills the role served by the alphanumeric insertion phrase in Subalgol. A LIST, however, is somewhat more elegant than a Subalgol INPUT or OUTPUT list since a LIST can be used for either input or output (i.e., as argument of either PUT or GET), and includes the types of its elements, therefore needing no accompanying FORMAT (which concept therefore fails to exist in BLINK).
- L. PUT,GET : These are simply procedures (always in core) which can have any number of LISTS as program reference parameters.
- M. RESTART : When several BLINK programs are to be compiled during the same 7090 run, the non-last programs use RESTART to terminate compilation rather than FINISH. RESTART reloads the BLINK compiler instead of returning control to the monitor.
- N. *GETCOR<integer><identifier> : This is a control card recognized by BLINK. *GETCOR is similar to RESTART, except that after completing the compiler output, the indicated disc file (rather than the compiler) is loaded.

- O. EXTERNAL PROCEDURE, EXTERNAL SUBROUTINE : These declarations, identical to those in Subalgol, allow linkages to be created on the LINC between BLINK subprograms and subprograms created by means other than the BLINK compiler.

II.

- A. Unlike Subalgol, BLINK has reserved variables. I1 through I7 (index registers), are simple variables of type integer with absolute address 1 through 7, respectively. These variables are GLOBAL and their values are not restored after an overlay.
- B. M, MF, and MH are GLOBAL arrays with absolute base address of 0. Their types are integer, floating, and half-word, respectively. These are used to great advantage together with I1 through I7:

M(I1) = M(I2)\$ results in the elegant code

LDA 2, STA 1

- C. I1I, ... , I7I are used in conjunction with the M() and MH() arrays in order to reference consecutive words, or half-words, of core. As an example, the following statements replace the contents of quarter 5 by the contents of quarter 4:

I1 = "3777" ; I2 = "2377" ;

REPEAT "400" ; M(I2I) = M(I1I) ;

COMMENT LDAI1 , STAI2 ;

Thus the value of the indicated index register is incremented before it is used as a subscript. When the above program is completed, I1 will contain "2377" and I2 will contain "2777" . (It is a quirk of the LINC that the core is logically divided into halves; thus 1777 is the predecessor of 2000 and 1777 is the predecessor of 0.)

In the case of half-words, index registers are incremented by 4000 rather than by 1. Thus if an index register were stepping through the characters of quarter 4, it would assume successively the values 2000, 6000, 2001, 6001, 2002, etc. The 4000-bit indicates the right-half of the word:

An index register can be made to point at a variable by a statement of the form `InI=<variable>` . The code generated is:

```
SET i n
```

```
<variable location>
```

The following program places the characters of the STRING S() into quarter 7, putting one character (right justified) into each word.

```
STRING S(20)=( alpha string );
```

```
I7I="3377";
```

```
COMMENT: There is canonical correspondnece between  
registers and quarters.
```

```
I2I=S(0); I2=I2+"4000";
```

```
REPEAT 40; M(I7I)=MH(I2I);
```

```
COMMENT: LDHi2 , STAi7 $
```

D. POINTER is that cell ("155") in QUARTER 0 which points to the top of the BUFFER. The statement `POINTER=M(POINTER)` would erase one record from the BUFFER. If we assume that the top record of the BUFFER begins with an alphabetic item, then the statement:

```
I2=M(POINTER)+"4001";
```

would allow `MN(I2I)` to reference successive characters of that first item.

APPENDIX 1: QUARTER O

"/" indicates data location

0000 16	0100 STC 16	0200 STC 210	0300 SET+11
0001 CLR	0101 LDA+16	0201 ADD 17	0301 0
0002 STA	0102 SCR 6	0202 STA 11	0302 LDA+11
0003 3140	0103 STC 15	0203 LDA+13	0303 HSE+
0004 STA	0104 LDA 16	0204 STA+11	0304 6440
0005 3000	0105 RCL+	0205 XSK+17	0305 COM
0006 RCG	0106 7700	0206 JMP 203	0306 ADD 11
0007 4011	0107 ADD 117	0207 LDA+	0307 STC 301
0010 JMP 400	0110 ADD 17	0210 0	0310 LDA+11
0011 HLT	0111 JMP 113	0211 STA+11	0311 STA+13-
0012 HLT	0112 LDA+17	0212 JMP 160	0312 XSK+17
0013 HLT	0113 STA+16	0213 JMP 231	0313 JMP 310
0014 HLT	0114 XSK+15	GET 0214 LDA	0314 JMP 231
0015 HLT	0115 JMP 112	0215 0	0315 HLT
0016 HLT	0116 LDA+	0216 STC 176	0316 HLT
0017 HLT	0117 6001	0217 JMP 20	0317 HLT
SAVE 0020 SET+15	0120 ADD 16	/0220 STC 301	0320 HLT
0021 3000	0121 STC 122	0221 JMP 154	0321 HLT
0022 LDA+	0122 JMP 0	0222 AZE+	0322 HLT
0023 1776 REPEAT	0123 LDA	EMPTY 0223 HLT	0323 HLT
0024 ADD 0	0124 0	0224 STC 301	0324 HLT
0025 STA+15	0125 STC 210	0225 JMP 175	0325 HLT
0026 STC 16	0126 JMP 20	LST.OP 0226 STC 243	0326 HLT
0027 LDA+16	?/0127 SAM+15	0227 ADD 0	0327 HLT
0030 ROL+1	/0130 HLT 10	0230 STC 271	0330 HLT
0031 SCR 1	/0131 STC 136	0231 JMP 20	0331 HLT
0032 STC 14	0132 ADD 210	/0232 ROL 3	0332 HLT
0033 LDA 14	0133 JMP 74	/0233 STC 276	0333 HLT
0034 STA+15	/0134 JMP 1502	0234 JMP 271	0334 HLT
0035 LZE+	/0135 HLT	GET.CL 0235 ADD 0	0335 HLT
0036 JMP 27	/0136 HLT	0236 STC 243 RETURN	0336 JMP 46
0037 LDA	0137 LDA+17	0237 JMP 154	0337 STC 341
0040 21	0140 AZE	0240 STC 155	0340 RCG
0041 STA+15	0141 APO	LST.CL 0241 JMP 46	0341 0
0042 LDA	0142 JMP 150	0242 CLR	0342 JMP 0
0043 15	0143 COM	0243 JMP 0 OVERLAY	0343 AZE+
0044 STC 21	0144 STC 10	STRING 0244 ADD 0	0344 JMP 336
0045 JMP 116	0145 JMP 135	0245 JMP 74	0345 APO
RESTORE 0046 LDA	0146 XSK+10	/0246 JMP 1601	0346 JMP 363-
0047 0	0147 JMP 135	/0247 HLT	0347 STC 17
0050 STC 73	EXIT 0150 LDA	0250 LDA+17	0350 ADD 0
0051 SET 15	0151 136	0251 STC 13	0351 STC 342
0052 21	0152 STC 243	0252 ADD 247	0352 JMP 20
0053 LDA 15	0153 JMP 241	0253 STC 271	/0353 SCR 2
0054 AZE+	0154 SET+11	0254 JMP 46	/0354 STC 356
EMPTY 0055 HLT	POINTER 0155 3140	0255 LDA 13	0355 WCG+
0056 STA	0156 LDA 11	0256 COM	0356 4101
0057 21	0157 JMP 0	0257 SRO	0357 ADD 130
0060 STC 15	0160 LDA	0260 276	0360 ADD 356
0061 LDA+15	0161 11	0261 ADD 264	0361 STC 356
0062 STC 14	0162 STC 155	0262 JMP 274	0362 LDA
0063 LDA+14	0163 JMP 0	LST.EL 0263 ADA+	0363 17
0064 ROL+1	PUT 0164 LDA	0264 7776	0364 ADA+
0065 SCR 1	0165 0	0265 STC 13	0365 7770
0066 STC 16	0166 STC 176	0266 ADD 0	0366 APO+
0067 LDA+15	0167 JMP 154	0267 JMP 74	0367 CLR
0070 STA 16	0170 LDA	/0270 JMP 1601	0370 ADA+
0071 LZE+	0171 11	/0271 HLT	0371 7
0072 JMP 63	0172 STA+11	/0272 JMP 46	0372 ROL 3
0073 JMP 0	0173 JMP 160	0273 LDA+17	0373 ADA+
PARAM 0074 ADD 23	0174 COM	0274 STC 17	0374 4001
0075 STC 17	0175 STC 276	0275 SRO+	0375 STC 377-
0076 ADD 0	0176 JMP 0	0276 0	0376 0

APPENDIX II: LOSS Character Codes.

<u>CHARACTER</u>	<u>CODE</u>	<u>CHARACTER</u>	<u>CODE</u>
	00	A	41
!	01	B	42
"	02	C	43
#	03	D	44
\$	04	E	45
%	05	F	46
&	06	G	47
'	07	H	50
(10	I	51
)	11	J	52
*	12	K	53
+	13	L	54
,	14	M	55
-	15	N	56
.	16	O	57
/	17	P	60
0	20	Q	61
1	21	R	62
2	22	S	63
3	23	T	64
4	24	U	65
5	25	V	66
6	26	W	67
7	27	X	70
8	30	Y	71
9	31	Z	72
:	32	(carr. ret.)	73
;	33	(end of text)	74
<	34		
=	35		
>	36		
?	37		
@	40		

code derivation:

LDA '
 (teletype code)
 COM
 ADA 1
 0277
 SCR 1

Appendix III: A Detailed Description of LOSS, Especially QUARTER 0.

SAVE AND RESTORE:

These routines govern a push-down stack whose presence allows the other routines of QUARTER 0 to be recursive.

```
JMP SAVE
LOCATION1
LOCATION2
.
.
LOCATIONn+4000
```

causes the n locations together with their contents to be saved on the top of the push-down stack. The call `JMP RESTORE` causes the topmost list of locations on the stack to be restored to their former contents. This push-down stack occupies cells 3000 to 3140 and is called the SAVE-BUFFER, as opposed to the PUT-BUFFER which begins at 3140.

PARAMS:

Consider the BALGOL statement:

`P(3,YZL1,L2) . . .` (a procedure call)

which would be equivalent to the following LINC code:

```
LDA
Y
STC*+3
JMP P
0003
0000
Z
JMP L1
JMP L2
```

Value parameters are thus represented by their values in the calling sequence, name parameters by their addresses, and program reference parameters are preceded by the JMP prefix.

The heading of procedure P() might appear as follows:

```

P: LDA
    0000
    JMP PARAMS
    7405
R: 0000
    0000
    0000
    LDAi17
        STA list for Z
    LDAi17 }
        STA list for L1
    etc.  }
```

Where 7405 derives from the formula $100(76 - \text{no. of value params}) + (\text{no. of params})$, R will be assigned the return address for each call of P(), and R+1 and R+2 will be assigned the values of the two value parameters. Index register 17 is left by the PARAMS routine so that the non-value parameters can be conveniently obtained and stored where required in the body of P().

REPEAT:

The program

```

                JMP PRPEAT
                JMP PAST
                0005
                } code

                JMP STEP

PAST: etc.
```

causes "code" to be executed 5 times. REPEAT is completely recursive (i.e., many REPEAT loops may be nested), and is the sole user of index register 10. A REPEAT loop can be terminated only by completing the full number of iterations, or alternatively, by executing the instruction JMP EXIT within the loop. If the count parameter (5 in the above example) is zero or negative, the loop

is not executed at all.

PUT and GET:

These can best be explained through an example. The following program will replace each item in the LIST L2 by the corresponding item in the LIST L1. A,B,X,Y, are variables and S1() and S2() are strings. First, the Balgol statements:

```
STRING S1(5)=('ALPHAS'),S2(5);
LIST L1(A,B,S1()),L2(X,Y,S2());
PUT(;;L1);GET(;;L2);
etc.
```

Next, the corresponding LINC code:

```

      JMP PUT
      JMP L1
      JMP GET
      JMP L2
      JMP GET.CL
      etc.
L1:ADD 0
      JMP LST.OP
      LDAi
      A
      JMP LST.EL
3776 . . . (control word; the first digit is type: 3 is
      LDAi integer, 7 alphabetic, and 1 floating. The
      B next 3 digits contain the complement of the
      JMP LST.EL size of the item.)
3776
      JMP STRING
      S1
      JMP LST.CL
L2:ADD 0
      etc....
      JMP LST.CL
S1:0005 (string size)
      4154
      6050
      4163
      7474 (end code)
      0000
      7474 ('extra margin' end code)
S2:0005
      0000
      0000
      0000
      0000
      0000
      7474
```


The BUFFER:

The BUFFER has a linked-list structure, the top of which is POINTER (cell 155). If in the previous example we assume that A and B have the values 7 and 24, respectively, then after the statement PUT(;;L1) , the BUFFER would have the following appearance:

LOCATION/CONTENTS

	0155	3154	
	3140	0000	. . . null contents denote BUFFER bottom
	3141	3776	
	3142	0007	
	3143	3776	
	3144	0024	
	3145	7771	
	3146	4154	
	3147	6050	
	3150	4163	
	3151	7474	
	3152	0000	
	3153	7474	
	3154	3140	

If GET is ever entered when the BUFFER is empty, i.e., when location 155 contains 3140, then a halt occurs at location 223; if RESTORE is called when the SAVE-BUFFER is empty, a halt occurs at location 55.

OVERLAY:

When it is desired to OVERLAY a program from the stack, the program number is loaded into the accumulator, and JMP OVERLAY is executed. When a program has completed its function and wishes to return to its caller, JMP RETURN is executed. OVERLAY 0 is equivalent to RETURN. The sequence

```

      (case 1) LDAi
              0005
      JMP OVERLAY
      JMP RETURN

```

is much more efficiently accomplished by:

```

(case II) LDAi
          7772 (i.e., -5)
          JMP OVERLAY

```

for in the second case, the present contents of core are neither written on tape nor read back in when program 5 is finished; rather program 5's RETURN is placed on the same level with the RETURN appearing in case 1.

If an argument of OVERLAY is greater than the size of the program stack, then the last program on the stack is loaded; thus a copy of the MONITOR is usually in both the first and last positions. If RETURN is called when the OVERLAY stack is empty, a halt occurs at location 55 (since RESTORE will be spuriously called).

TEXTS:

The first two words of the 0th block (the index) of each BOOK are

```

      block no.   (0,100, or 200, etc.)
      4253        ('BK')

```

Each succeeding group of four words are TEXT entries

char ₁	/	char ₂
char ₃	/	char ₄
char ₅	/	size (i.e., length in blocks)
initial block		

The end of the index is denoted by a zero where the first two characters of the next name would be.

Each text is headed by the following four word code:

char ₁	/	char ₂
char ₃	/	char ₄
char ₅	/	max. size
current size/ type		

The purpose of having types is for file protection; when some program is written which will create a special kind of TEXTS - it can, of course, use

use any of the 100 available types. Thus far type 0 denotes a standard alphabetic TEXT (using the half word codes in Appendix II), and type 41 ('A') denotes an 'absolute' TEXT, i.e., a TEXT whose first block consist of a header alone and whose next 5 blocks are an absolute program ready to be placed on the program stack.

Appendix IV: How to Run a BLINK Program.

The BLINK3 compiler is stored in the disc files of Stanford's 7090 computer. In order to use this compiler, it is necessary to prepare a card deck as follows:

<u>No. 1 Card:</u>	SYSTEM	:	F-INFO
	TAPES	:	Mount on A3, at <u>low density</u> , a tape which can be removed from the Computation Center.
<u>No. 2 Card:</u>	SYSTEM	:	F-INFO
<u>Control Card:</u>	Cols. 1-6	:	BLINK3 file number (changes periodically), right justified.
	Cols. 7-11:		BLINK

There should follow a BLINK source deck. This deck should be terminated by a RESTART, FINISH, or GETCOR card. If a RESTART terminator is used, it should be followed by another BLINK program.

The output produced by BLINK3 will be on the tape which was mounted on unit A3. This output is quite similar to that produced by the SUBALGOL compiler, i.e., listings of the source decks, diagnostic messages, symbol tables of the compiled programs (these being especially useful for console debugging). In addition, however, the tape will contain the actual LINC code produced by the compiler.

If no compiler error messages are produced, the tape is brought over to the LINC, mounted on the LINC's tape unit, and read by an appropriate LINC program. One such program merely searches the tape, ignoring all that it sees, until it comes to compiled code. That code is then transferred to the LINC tape, unit 1, in the form of a TEXT.

One of the principle drawbacks of the BLINK3 system is the means by which information is transferred from the 7090 to the LINC. Not only is it incon-

venient to have to physically travel to the Computation Center, but jobs requiring special tape handling are not given top scheduling priority on the 7090.

The BLINK4 system will employ an electrical connection between the LINC and the Computation Center's PDP. The BLINK programmer, instead of preparing a card deck at the Computation Center, will prepare a TEXT on the LINC. When completed, this TEXT will be sent to the 7090 via the PDP. The first line of the TEXT will actually be the No. 2 card expected by the 7090 monitor. Instead of using tape A3, the BLINK4 compiler will send its output directly to the PDP, which will relay it to the LINC. The LINC in turn will write the output on IBM tape. The tape can be examined on the LINC's scope and never need be listed. If the tape contains error messages, then it is only necessary to alter the original TEXT and re-send it to the 7090. If there are no error messages, then the procedure becomes the same as under BLINK3.

1 1

2 2

3 3

4 4

5 5

6 6

7 7

8 8

9 9

10 10

11 11

12 12

13 13

THE LINC TELETYPE MONITOR SYSTEM

/ LINCT

The System consists of a monitor which accepts Macro-instructions and associated octal parameters from the teletype and separately coded programs which are executed to achieve the desired result and return to the monitor. The requirements to use LINCT are a standard LINC and a Teletype Corporation series 33 teletype attached to relay #0 and External Line #0. Tape requirements are Blocks 200 and forward on unit 0, as the system is followed by an indefinite scratch area a practical upper limit of 277 is satisfactory.

The monitor is started by an 0700 0200 in the switches and a START 20. A return, line feed is the signal that the monitor is ready to accept input. The operator then types a 2 letter Macro code followed by the appropriate octal parameters and unit numbers (binary only). Commas separate fields and blanks are ignored. All parameters need not be explicitly specified as they are initially defined as zero; however, as unit 1 is desirable as a library tape, unit numbers are assumed as 1 unless a 0 is typed in the field (in some cases a , is necessary to "open" the field, but once a field is opened 1 is assumed, unless zero is typed, ,, means 1). An error in a calling sequence (e.g. illegal macro code, something besides 0 or 1 in a unit field, a non-octal digit in a octal field, or too many parameters in some cases) will result in a NO being typed back followed by a carriage return, line feed signifying ready status for a new line. The RUB OUT key is interpreted as an illegal character resulting in the NO and may be used to delete the line. The RETURN key effects execution of the Macro.

The following pages contain write ups of the Macros with descriptions and calling sequences. Also a page of actual teletype operation.

1. Input Type: IN n,u,x

This program receives alphanumeric text from the teletype and record it on tape in successive blocks beginning at Block n, Unit u (initially assumed 1).

Input Description: All alphabetic, numeric, and special characters are valid except ? which is ignored. The RUB OUT key will delete only the line currently being typed (multiple depressions have no effect on the text). Upon depressing it the program will do a carriage return, type ? X's over the junk that is deleted and proceed to a new line.

To end a line, press RETURN. The program gives the line feed when ready to accept a new line (usually immediate, but delayed when writing tape.) To terminate input, press EOT (CTRL & D keys) immediately following a RETURN, at any other place a NO is typed back and the EOT is ignored. The program after an EOT will write out the remaining text and type the message: LAST BLOCK USED IS ??? for tape logging purposes. Control is then returned to the monitor.

Line Numbering: If $x \neq 0$ numbering is suppressed. If $x=0$ (normal) an octal line number for the preceding line will be typed every eighth line beginning after line zero. The number is preceded by \times to avoid confusion with the numerous 4digit numbers that appear.

Output Format: The first two words of every block are 7575_8 , the third word is negative if the block is last in the text (never looked at in the system but might be of value). The text begins in the left half of the fourth word and continues by LINC half word indexing through the entire block. The end code is signaled by a 13_8 following a 12_8 (EOL code). This places the restriction that the character \ cannot be first in the line.

2. Type (list) Type: TY n,u,L₁,L₂,N_c,x

Type will list the text beginning at Block n, Unit u (initially assumed 1) under control of the remaining parameters.

Normal Format: If L₁=L₂=N_c=x=0 the printed output of the entire text has the same format as the input listing except for deleted lines.

Unusual Formats, Control Parameters:

L₁ : Begins printing at line number equal to L₁

L₂ : Stops printing at line number equal to L₂ ;however if L₂=0 the entire text after L₁ is printed. Either L may be greater than the last line number meaning equality to it.

N_c : Prints the first N characters per line. Useful for quick and dirty listings to get line numbers after alterations.

x ; x≠0 suppresses line numbering.

Notes: If a block read does not have the 7575 text code a NO is typed and immediate return to the monitor is made. If the line is longer than 65₁₀ characters, the program will start a new line on the teletype and continue printing the same line of text.

3. Group Type: GR N₁,U₁,N₂,U₂,N₃,U₃,.....N_n,U_n

Group will group the n texts at Block N₁, Unit U₁ into one text and store the result at the System scratch area (around 240 depending on the edition being used) on unit 0. The main purpose of this program is to prepare multiple texts for assembly. From 0 to 174₈ texts may be called for. Grouping is equivalent to catenation, i.e. the first of text I follows the last line of text I-1 and the last line of text I is followed by the first line of text I+1. Text 1 follows nothing and text n is followed by the text end code.

Operating Notes: A NO is typed if any block read lacks the text code.

At the end of the grouping the message :m m BLOCKS GROUPED AT ???

is written before return to the monitor. m is the number of blocks

written at the beginning of the scratch area which is given in the ???.

Caution: Only a one block buffer is used so nothing may be inserted in front of the scratch area text but may be appended.

4. Copy Type: CP m,n₁,u₁,n₂,u₂

The program will copy m blocks from block n₁, unit u₁ to block n₂, unit u₂. If m>1, successive blocks are copied. The information may be of any type and is not limited to texts.

Caution: A three block buffer is used so if moving more than three blocks forward on the same unit care must be taken to avoid clobbering blocks which have yet to be read. The range of m is 0 to 1000.

5. Alter Type: AL n,u,x

This program will perform a group of insertions and deletions to the text at block n, unit u. It makes use of the scratch area and programs Input and Copy. A brief description of its operation is in order. First the macro is typed, then the monitor instructs Input to place the alteration text at the beginning of the scratch area. The Alteration Text is then typed in (Format described below).ended with an EOT (no last block message is typed). Input then enter Alter. Alter reads the Alteration text and the text to be altered (n,u). It writes the altered text in the scratch area but immediately following the Alteration Text (Note: The altered text is never at the beginning of the scratch area). It then types a message and conditionally reenters Copy to return the altered text to block n,u. In any case the monitor is re-entered.

Alteration Text: Alteration instruction lines and lines to be inserted in the text make up the Alteration Text. The alteration instructions refer to line numbers in the text to be altered and references must be in sequence from line 0 forward. The format of alteration instructions is; /m,n_{return} no blanks are permitted on the line.

The program will remove lines from the beginning of line m to the beginning of line n and will insert any lines of text that follow it until another alteration instruction is encountered. (or an end of text)

Note: A slash cannot be the first character in the Alteration text unless the line is an alteration instruction (no restriction on original text). The message ILLEGAL PROCEDURE is typed if: a block is read which does not contain the 7575 code, an alteration instruction of an illegal format, or an alteration instruction of legal format but where $m > n$, $m < (\text{the previous instructions } n)$, or $m > (\text{last line number of the original text})$.

As the process is a merge, line numbering of the old text is preserved

throughout the one pass, after completion however, the line numbering is dependent on the alterations made and a partial print may be used to determine line numbering in places of interest.

If $x=0$ the text will be returned in place of the original text if the length of the altered text is less than or equal to the original, otherwise it will be left in the scratch area and the message `n BLOCKS REMAIN AT x` will be typed, where n is the number of blocks and x is the location of the first block. If the altered text is returned the message `n BLOCKS RETURNED` will appear. The purpose of this criterion is to prevent clobbering a block of text immediately following the original text. If $x=1$ the text is unconditionally returned, and if $x=2$ the text is not returned.

An example is in order:

AL356 means alter block 356, unit 1, $x=0$
/1,1 says "remove nothing and insert the following lines in front of line 1"
ALPHA
BETA
GAMMA these three lines are inserted
/5,10 says "remove lines 5,6,7, and insert" but there is nothing to insert.
/12,13 says "remove lines 12 and insert before 13"
DELTA this line is substituted for 12
/17,100 presuming a text of say 62 lines, this will remove line 17 through
EPSILON the end of the text and append whatever follows it to an end of text
ZETA (another alteration instruction is illegal as m must be less than
ETA or equal to 62 and greater than or equal to 100)
THETA
(eot) whereupon the alteration is performed the text is found to be smaller
and blocks are returned, the message is then typed.
1 BLOCKS RETURNED.

A note on Grouping: Grouping is an easier way to insert information before an existing text or appending to it. As an altered text is never left at the beginning of a scratch block it is necessary to group it to place it at the beginning (one block of text may be grouped in front of it safely).

6. Assemble. Type: AS

LINCT has been tied into the LAP Convert Metacommand, which works quite satisfactorily, through a program which transforms LINCT text at the beginning of the system scratch area into LAP text and places the result in blocks 336 forward (LAP input area) and enters the LAP converter which assembles to blocks 330 333 (270 to 277 for 2K version). The rules of line structure given in the LAP III Manual must be adhered to, obviously a new special character set is necessary as LAP uses a somewhat unusual set. The changed characters were chosen for typing convenience and resemblance was a secondary situation. The following is the list of changes

LAP	LINCT
i	; (semi colon)
p	* (asterisk)
u	! (exclamation mark)
l	/ (slash)
Origin	\$ (dollar sign)
Tag	: (colon)

OPERATIONAL SAMPLE FROM TELETYPE

```
IN523,1
$220
LDA;
1777
ROL 3
:5H JMP 7B
WRC;10 (note: unit 1 if from cards)
2/240
RDC ;!
4A
4A=230
JMP *-5
LAST BLOCK USED IS 523
GR523
AS
```

7. Execute Type: XE n,n,n,....,n

This is not a program but a means of loading and executing a program. The monitor will place the first parameter in location 1375 and successive locations thereafter, when RETURN is pressed the monitor jumps to 1375 and the octal commands typed in will be Executed. The reason 1375 was chosen was that one may do an RDC and a JMP and it will leave parameters in 1400 forward or if a program is to be read into quarter 2, three 16's (NOP) may be given and instructions are in quarter 3. Overlaying is a hit and miss proposition as a ^{Missed} tape check on the first attempt to read will cause error.

A Floating Point Subroutine Package for the LINC
Jeremy Pool

This package was written for programs compiled by "Blink", an IBM 7090 Balgol compiler for the Linc, written by Richard Moore; however, it is completely compatible with any machine language program.

The arithmetic routines - add, multiply, divide - and the float subroutine are, with minor alterations, those written by J.C. Dill, W. M. Stauffer, and R. W. Stacy of the University of North Carolina.

In this package the format for floating point numbers is a one word exponent followed by a one word mantissa. Both words are signed, one's complement numbers (standard form for the Linc). Zero is designated by a zero exponent and a zero mantissa. Floating point numbers must be in standard form, so that the mantissa has an absolute value between $010\ 000\ 000\ 000$ and $011\ 111\ 111\ 111$. The decimal point is understood to be between bits 11 and 10 of the mantissa.

In addressing it is always the first word, the exponent, which is specified.

The calling sequence is as follows:

```
JMP    400
      A1
      01
      A2
      02
      .
      .
      .
      An
      On
      Next instruction
```

A1 is the address of the first operand. Three possible formats for this address are possible:

```
A1 > 0 ; A1 = absolute address of operand
A1 = 0  ; The operand is the floating accumulator
A1 < 0  ; A1 = indirect address of operand
```

For indirect addressing, the address is not complemented; only the 11 bit must be set to 1. Thus with A1 = 4063, location 63 contains the address of the operand, not location 3714.

01 is the desired operation. Two forms are possible:

01 < 0 ; Execute the specified subroutine, and then continue to execute the next specified subroutine.

01 > 0 Execute the specified subroutine, which is the last in the series of subroutines, and return and execute the next instruction in location p + 1.

Here again, when 01 < 0, this is specified by setting to one the 11 bit, not by complementing the entire number. Thus 4002 means add and continue to execute floating point instructions while 0002 means add and return from the floating point package.

Some of the routines are "integer" subroutines and assume one of the numbers involved to be an integer. In this case the actual address of the integer is specified by the operand, directly or indirectly.

The subroutine codes, their mnemonics, and their explanations are as follows:

(op = operand; FAC = floating accumulator; ac = Linc's accumulator)

1. CLA Clear and add $c(op) \longrightarrow c(FAC)$
2. ADD Add $c(op) + c(FAC) \longrightarrow c(FAC)$
3. COM Complement complement of $c(op) \longrightarrow c(FAC)$
4. MUL Multiply $c(op) \times c(FAC) \longrightarrow c(FAC)$
5. DFA Divide (a) $c(FAC) / c(op) \longrightarrow c(FAC)$
6. DAF Divide (b) $c(op) / c(FAC) \longrightarrow c(FAC)$
7. IAD Integer Add $c(op) + c(FAC) \longrightarrow c(FAC)$; $c(op) = I$

In the previous subroutine and in some of the following, the operand is assumed to be an integer (I).

10. IML Integer multiply $c(op) \times c(FAC) \longrightarrow c(FAC)$; $c(op) = I$
11. DFI Integer divide (a) $c(FAC) / c(op) \longrightarrow c(FAC)$; $c(op) = I$
12. DIF Integer divide (b) $c(op) / c(FAC) \longrightarrow c(FAC)$; $c(op) = I$
13. FIX Fix $c(op)$ is converted to a fixed point number (an integer), and is stored in the regular, Linc, accumulator. Numbers are not rounded; all fractional parts are lost. Any number less than one is stored as zero. Any number greater than $3777_{(8)}$ or less than $-3777_{(8)}$ is converted to 3777 or -3777 respectively.
14. FIT Float $c(op)$ is assumed to be an integer. It is converted to a floating point number and replaces $c(FAC)$.
15. CLR Clear Storage $0 \longrightarrow c(op)$
16. MAX Maximum The $c(op)$ is compared with $c(FAC)$. The larger value replaces $c(FAC)$.
17. MIN Minimum The $c(op)$ is compared with $c(FAC)$. The smaller value replaces $c(FAC)$.
20. SGN Sign If $c(op) < 0$, then $-1 \longrightarrow c(ac)$
If $c(op) = 0$, then $0 \longrightarrow c(ac)$
If $c(op) > 0$, then $1 \longrightarrow c(ac)$.

21. INC Increment $c(op) + c(FAC) \longrightarrow c(FAC)$ and $\longrightarrow c(op)$. This is the floating point counterpart of Linck's add to memory instruction.
22. IIN Integer Increment $c(op) + c(FAC) \longrightarrow c(FAC)$ and $c(op)$; $c(FAC) = 1$
Note that in this instruction it is the FAC, not the operand, which is assumed to be an integer.
23. STO Store $c(FAC) \longrightarrow c(op)$
24. SSP Set Sign Plus $|c(op)| \longrightarrow c(FAC)$
25. SSM Set Sign Minus $-|c(op)| \longrightarrow c(FAC)$

26. SQT Square root ; $\sqrt{|op|} \longrightarrow FAC$

27. IPT Input ; the number inputted on the keyboard $\longrightarrow op$
The number is inputted in decimal and is terminated by a space.
The number may be preceded by a minus sign. Any of the following inputs are allowable:

27.345 \triangle
-.0001 \triangle
996 \triangle
-101 \triangle
-62. \triangle
 \triangle (=0)

There is no limit to the number of digits inputted. Pressing "del" at any time during an input deletes what has been entered and the entire number must be retyped.

30. OPT Output ; the operand is outputted on the teletype in the following format:

X.XXX, XXX return and line feed

The first four digits are the decimal mantissa and the last three the characteristic as a power of ten.

Also $PKG = JMP400$. The mnemonics are used in an assembly program to be described.

If locations 1472 and 3742 are altered so that they both hold "4276", the teletype does not return after it has outputted a number; it spaces once. (Normally these locations hold "6570")

The actual teletype output routine is included as a subroutine within the package, so that it can be jumped to from outside the subroutine package. To type a character, load the accumulator with that character's teletype code and jump to location 1742. Control will automatically be returned to $p + 1$. Index registers 12 and 15 are used by this subroutine and are not restored if one jumps to 1742. A modification is included for scope output of the same format.

The package occupies all of quarters one, two, and three. Quarter 7 is used from location 3700 to 3756. All index registers are restored to their previous

value except in the case mentioned above.

The floating accumulator is locations 1120 and 1121.

No error detection is provided. Overflow of exponents in arithmetic subroutines will yield incorrect answers, not error messages. The same is true of invalid operation codes, etc.

A sample program which calculates $\frac{3}{x^2-3x}$, which stores the result in the

floating accumulator, and which leaves +1, -1, or 0 in Linc's accumulator, depending upon the value of the result, follows:

x is in 1G
 3 (integer) is in 1T
 locations 24 and following contain

```

                                LDA
                                0
                                JMP 1000

```

JMP 24

1T
 4014

0
 4003

} Loads -3, floating point, into the FAC

1G
 4002

} $x-3 = c(\text{FAC})$

1G
 4004

} $x(x-3) = x^2-3x = c(\text{FAC})$

1T
 4012

} $\frac{3}{x^2-3x} = c(\text{FAC})$

0
 0020

} Determines sign of answer Exit because the 11 bit = 0

Next instruction

Program follower

To use:

Read in program to be tested and execute it once.

Then read it out temporarily on tape and read it back into memory, in executed form, into blocks of upper core corresponding to those blocks of lower core where the program normally operates, i.e., quarter 0 into quarter 4, 1 into 5, etc.

On sense switches set the quarters which the program uses (actual lower core quarters).

On the right switches set the address of the first executed instruction.

Read in the Program follower and start 20.

When the program halts locations 20 and following will contain the locations of your program which are instructions. The following is an example of the final output:

<u>loc.</u>	<u>contents</u>	
20	20	This means instructions were contained in locations 20 through 176 and 405 through 760 of your program.
21	176	
22	405	
23	760	
24	0	
25	0	
26	0	

The program follower is not perfect. It will not catch returns from subroutines where the return address is manipulated to be anything besides p+1 or a constant return address. It will not catch jumps executed by pulling addresses out of a jump table. It will assume that all XSK instructions can proceed to both p+1 and p+2, while this is not always true. Therefore, the results may contain a few locations

which are data and may omit locations which contain instructions.

The program follower is just that; it does not tell you what parts of your program were meant to be instructions, it tells you which locations can be reached, as instructions, by the various jumps and branches of your program. Thus it provides a good method for troubleshooting a program by showing you where your program actually can go.

Mnemonic dump

To use:

Read program to be typed into upper core in quarters corresponding to the lower core location of the program, i.e., a program which runs in quarters 0 and 2 should be read into quarters 4 and 6.

Have tape JP on unit 1.

Read B1 310 into quarter 0 and start 20.

Type on the kbd one-digit numbers corresponding to the quarters used by the program. For the case mentioned above, type 0 space 2.

Separate these digits by spaces.

Then type in the locations which are instructions in the form specified below.

If the program postulated above ran from 20 to 360 and from 1000 to 1377 the entire input should be as follows:

<u>0</u> Δ <u>2</u> Δ	<u>0020</u> Δ <u>0360</u> Δ <u>1000</u> Δ <u>1377</u> Δ	EOL
<u>Quarters</u>	<u>Instruction Locations</u>	
Location Ø may not be specified as an instruction location		

Sense switches control the output format as follows:

All set to Ø	=	single column output
SW 1 at 1	=	2-column output, numbering spaced by 200
SW 1, 3 at 1	=	2-column output, numbering spaced by 100
SW 1, 2, 3 up	=	4-column output

Appendix: B

Log Book

Boston, 1963,
July 13: The frame was received and a preliminary inspection located a few unsoldered wires which were corrected. The power supply was installed and power was applied to the frame with no cords in place. All voltages were checked and found to ~~be~~ present in all boxes with the exception of the ⁻¹⁵ marginal check voltage on the control side. This was found to be the result of a jumper not being present in the S box between the yellow power pin. The jumper was inserted. Power was checked on the fontail connector and found to be present. External ~~pos~~ resistors and capacitors were added to the frame as required.

July 14: The frame blower was installed. Cords were then installed in the frame and it was found that the top and bottom cover plates on K and Z boxes were inverted. These were corrected and the rest of the cards and the memory were installed. The console was connected to frame.

July 15: Wires were added to the frame as per change notice. They were: T20V-ground, T24V-ground, U23M-ground, U23P-Gnd, V15M-V18E, FT-17-ground. Power was applied to the frame and the -15V circuit breaker tripped. A wiring error was found on the I stop button on the console which put -15 directly to ground. This was corrected. Power was again applied. $\pm 18V$ and $-3V$ were readjusted. It was found to be impossible to adjust the -10 supply. It was found that the A-D ladder switches, with no signal were putting about 30 millivolts backwards into the -10 supply from the -15 supply. This was corrected by putting a 220 ohm resistor

across the -10 supply. The supply was then ~~then~~ adjusted to the proper value. The memory was then turned on per procedure without difficulty. The range between excess ones and no ones ranged from 9 to 15 turns on the sense amplifiers slicing level. It was discovered that there was no ~~data~~ Memory to C register transfer. This was due to lack of the enable level SA \rightarrow C. A jumper was found to be missing on the frame and was inserted. Full address instructions were checked and operated properly except that STC did not clear the right 5 bits of A. This was due to a bad connection in plug ~~was~~. Console functions were checked and found to be correct except that Auto, Φ stop and XOE stop came on in a random manner. This was apparently due to random noise pick-up on the push-~~on~~ button lines. It was also found that Instruction by Instruction and cycle by cycle do not operate properly.

A keyboard program was inserted and found to operate except that the key-stroke signal from the keyboard does not always reach the computer. This is believed to be a design fault to be corrected.

July 16: A fix for "Crazy Auto" and friends was made by using R-C filter in 4110 package. These were inserted in the lines to the Console push buttons and no further difficulty has been encountered. The indicator unit was installed and found to work with only minimum adjustment. A square generating program was operated successfully.

July 17: The difficulty encountered in the I by I and C by C buttons was found to be due to the lack of the button pushed level from the console. This was traced to a missing jumper in the console which was inserted. Proper operation followed. Keyboard operation with the new unit is still giving trouble. Apparently no key-stroke signal gets through. The prototype unit works without difficulty. The difference is to be investigated.

July 18: The keyboard problem was found to be due to an improper relay sequencing. This was corrected by removing the key-stroke signal from the relay and generating it by a contact on the keyboard key-stroke bar. The A-D and DA ladders were tuned without difficulty and a good saw-tooth could be generated on the scope. No Modem failures have been noted to this point.

July 20. Tape deck was installed. It was found that the center taps of the head windings were not jumped from the left to the right plug. This is to be corrected. The right shoe of the right unit was found to be slightly narrow which caused a lifting of the tape from the head when running in the forward direction at high speed. With this exception, the unit works properly.

July 22. The analog input ladder was aligned. A difficulty was encountered with maintaining a zero setting on the sample and hold card. A new card was installed and the problem was corrected. The suspect card was just back in the machine and is being continuously operated to look for

Tape deck, there was erratic operation,
 but the trouble cleared in a few
 minutes. It is suspected that one of the
 protective delays may be marginal &
 requires a brief warm-up period. This
 will be watched closely for further
 failure.

Mortimer checked as follows

Box	10A	10B	-15	Program
K	0-15	0-15	-3 → -18	Tape
L	0-15	2-15	-4	"
M	0-15	0-15	-5	ADD & STL
N	0-15	0-15	-5	"
P	0-15	0-15	-5	"
R	0-15	0-15	-8.5 → -18	"
S	0-15	0-15	-5	Program & P.B.
T	0-15	0-15	-5 → -18	Tape Program
U	"	"	-4	"
V	"	"	-9	Display
W	"	"	-5	ADD-STL
X	"	"	-5	"
Y	+5.5	+6.5	-5	"
Z	0-15	0-15	None	"

Stanford
Aug 9, 63. The time arrived at Stanford
and was unloaded and checked for
physical damage. Everything seemed
in good condition.
Aug 10, 63 Cables cleaned and connected and
a test program run. Everything appears
to be working well.
Aug 20, 63 Memory was retuned without difficulty
and appears to be working properly.

Nov 15, 63 Tape Drive belt tension was adjusted
as per E.C. #3. No particular change in
operation has been noted. A few
apparently incorrect tape transfers have
been noted in fact but this may
be a program fault but it is
difficult to say for sure. This
fault cannot be observed in programs
involving simple tape manipulations.
Feb 12, 64 E.C. #7 and E.C. #8 made. No previous
faults involving the 4221's have been
noted. In fact, no machine faults have
been noted at all with the exception
of what appear to be a few random
incorrect tape transfers.

April 14, 64. Two output drivers Apr 11 and Apr 12
found faulty apparently due to inductive
spike pickup from the 026 keypunch
the 026 causes more trouble than it
is worth.

June 21, 1964, Tape unit heads and shoes
removed and surfaced. Burs were
definitely present on the shoes. ACP
delay check and found to be OK. Tape
units have given little trouble.

July 25 1964, New Mem address cards received
and installed. All wiring changes
per E.C. No. 12 made and upper
memory tuned and operated
without any problems. Mem
check program run and margin
found to be 15-145-27.